Заинтересовался идеей создания двухмерных лабиринтов, после того, как случайно набрел на ресурс <https://www.mazegenerator.net/>.

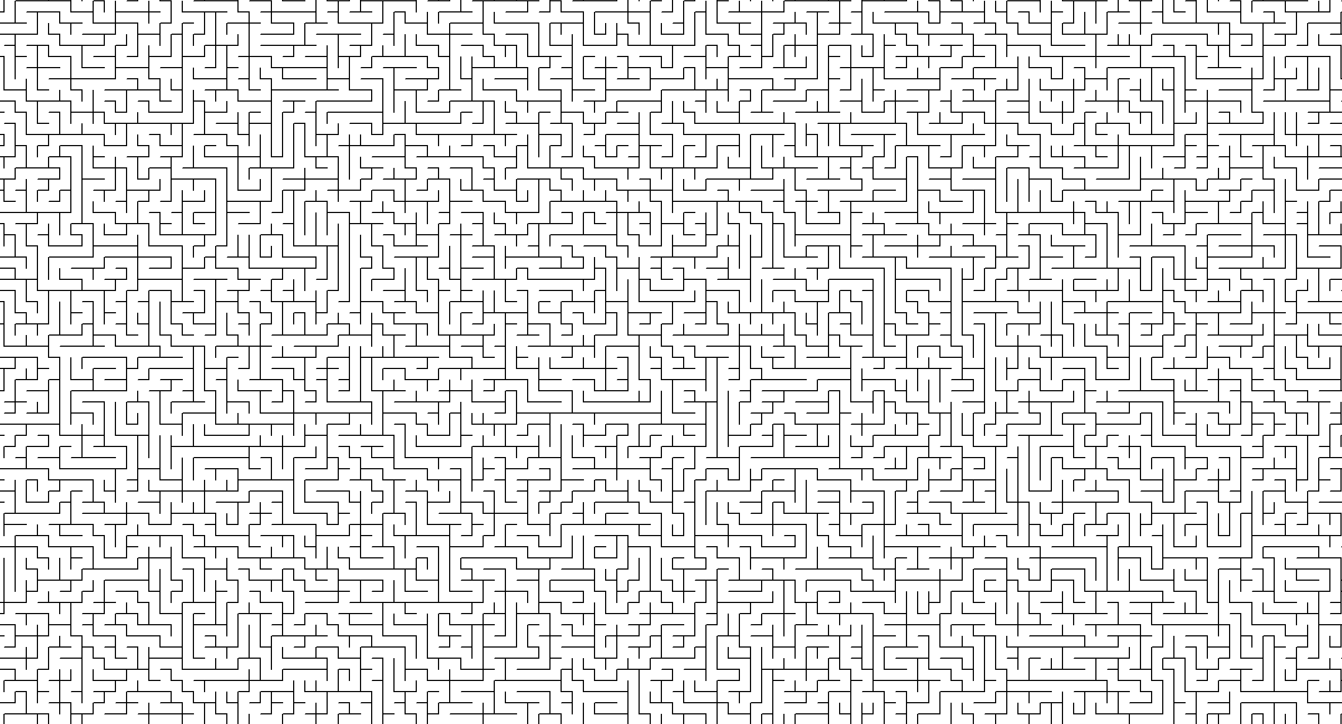
Полез изучать тему, нашел интересную статью на Хабре, где расписывались особенности типы и способы создания таких лабиринтов. К сожалению, представленный в ней материал оказался не очень понятным и наглядным, поэтому решил самостоятельно придумать алгоритм, способный быстро создавать лабиринт произвольного размера.

Насколько понял, подавляющее большинство существующих алгоритмов используют принципы рекурсивного поиска, с записью данных в стек, из которого они потом по мере необходимости берутся для возобновления этого самого поиска, в случае, когда трактор (условная машинка, которая грызет проход в монолите) попадает в тупик.

Мне такой подход крайне не понравился, и я придумал алгоритм, который осуществляет поиск таких путей за два прохода матрицы лабиринта. Возможно этот алгоритм уже кем-то придуман, но раньше мне такое на глаза не попадалось, поэтому смело называю его своим именем.

Итак, предлагаю вашему вниманию двухходовый алгоритм stalkself построения идеального прямоугольного лабиринта произвольной формы. Идеальный, в нашем случае означает, что две любых точки лабиринта можно соединить единственным путем. И соответственно к ним относятся точки входа и выхода.

Вот фрагмент получаемого лабиринта.



Важная особенность моего алгоритма – отсутствие привязки к этим точкам, поскольку для идеального лабиринта в любом случае сохраняется возможность найти единственный путь между ними.

Два этапа алгоритма:

Создание пространства путей в матрице за один ее проход

Объединение путей в единственный также за один проход

Программу писал на Паскале, поскольку это лучшее, что придумало человечество (ну или потому, что других языков не знаю), взяв за основу Эмбаркадеровскую Дельфи.

Бросил на форму два TEdit, чтобы указывать в них размерность матрицы лабиринта по Х и по Y. И компонент TImage для отрисовки лабиринта.

В первую очередь указываем тип матрицы:

type

cell = record

group:integer;

r,d:integer;

end;

И наши переменные:

var

form1: Tform1;

count:integer; //счетчик групп

cel:array of array of cell; //массив ячеек

x,y:integer;

Каждая ячейка в массиве cel имеет три поля:

group – номер группы

r и d – правая и нижняя границы соответственно (стенки лабиринта)

Создаем и заполняем массив после нажатия на кнопку:

x:=strtoint(edit1.Text); //Наши значения размера массива из эдитов

y:=strtoint(edit2.Text);

setlength(cel,x+1,y+1); //Задаем размеры массива ячеек с запасом +1, чтобы не возиться с нулевыми индексами

for j := 1 to y do //Заполняем пространство group нолями, а все стенки делаем закрытыми

for i := 1 to x do

begin

cel[i,j].group:=0;

cel[i,j].r:=1;

cel[i,j].d:=1;

end;

Все, теперь можем начинать работу алгоритма

1. Этап - создаем пространство путей, которые будут проложены трактором.

count:=1; // Наш счетчик номеров и количества групп

for j := 1 to y do //Первый проход матрицы по высоте и ширине

for I := 1 to x do

if cel[i,j].group=0 then //Находим пустую клетку и пускаем из нее трактор

begin

track(i,j);

inc(count); //Увеличиваем счетчик, после того, как трактор построил путь

end;

Собственно, код трактора:

procedure tform1.track(x1: Integer; y1: Integer);

var

k,mx,my:integer;

label start;

begin

//Процедура прохода и построения трактором лабиринта

mx:=x1;my:=y1;

cel[mx,my].group:=count; //присваиваем стартовой клетке номер новой группы

start:

k:=random(4)+1; //Случайное направление движения

case k of

1:

if (mx>1) then

if (cel[mx-1,my].group=0) then //Если слева пусто, идем туда и заодно убираем стенку

begin

cel[mx-1,my].r:=0;

cel[mx-1,my].group:=count;

mx:=mx-1;

goto start;

end;

2:

if (mx<x) then //Аналогично остальные направления

if (cel[mx+1,my].group=0) then

begin

cel[mx,my].r:=0;

cel[mx+1,my].group:=count;

mx:=mx+1;

goto start;

end;

3:

if (my>1) then

if (cel[mx,my-1].group=0) then

begin

cel[mx,my-1].d:=0;

cel[mx,my-1].group:=count;

my:=my-1;

goto start;

end;

4:

if (my<y) then

if (cel[mx,my+1].group=0) then

begin

cel[mx,my].d:=0;

cel[mx,my+1].group:=count;

my:=my+1;

goto start;

end;

end;

//Проверяем не тупик ли

if canmove(mx,my) then goto start;

end;

В конце проверка на возможность дальнейшего движения для трактора:

function tform1.canmove(x1: Integer; y1: Integer): LongBool;

begin

//Проверка на возможность хода

result:=false;

if x1>1 then if cel[x1-1,y1].group=0 then

begin

result:=true;exit;

end;

if x1<x then if cel[x1+1,y1].group=0 then

begin

result:=true;exit;

end;

if y1>1 then if cel[x1,y1-1].group=0 then

begin

result:=true;exit;

end;

if y1<y then if cel[x1,y1+1].group=0 then

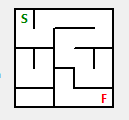
begin

result:=true;exit;

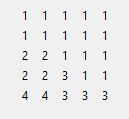
end;

end;

Все, наше пространство путей готово. Оно будет иметь примерно такой вид:



Соответствующая ему матрица групп:



Как видно из точки S в точку F дойти невозможно, тем не менее у нас уже есть четыре отдельных группы, каждая из которых представляет собой путь, пройденный трактором.

Чем более крупным является лабиринт, тем больше таких путей. Каждый из них является идеальным, поскольку изначально построен так, чтобы исключить вероятность создания петлей и замкнутых областей.

Осталось объединить их в один идеальный путь:

for i:= 1 to x do //Запускаем второй проход матрицы (первая ячейка всегда имеет номер группы 1)

for j:= 1 to y do

begin

if i<x then if cel[i+1,j].group<>1 then //Если находим группу справа от нас отличную от единицы то присоединяем к нашей единичной следующим образом:

begin

cel[i,j].r:=0; //Убираем заодно стенку

del(cel[i+1,j].group,i+1); //Меняем все найденные индексы группы на нашу единицу, для оптимизации процесса делаем это не с начала, а с ряда, где находится наша найденная клетка

end;

if j<y then if cel[i,j+1].group<>1 then //Аналогично смотрим клетку снизу

begin

cel[i,j].d:=0;

del(cel[i,j+1].group,i);

end;

end;

Процедура замены на единички:

procedure tform1.del(x1,x2: Integer); //x1 – наша группа, х2 – точка старта

var

i,j,f:integer;

begin

for i:= x2 to x do

begin

f:=0; //Флаг поиска групп в ряду для оптимизации

for j:= 1 to y do

if cel[i,j].group=x1 then

begin

cel[i,j].group:=1;

f:=1;

end;

if f=0 then exit; //Если не найдено групп в хотя бы в одном ряду, то в оставшихся рядах и искать нет смысла, чтобы не тратить ресурсы ПК

end;

end;

В результате получаем одну группу, которая соединяет все клетки лабиринта в один единственный путь.

Осталось только нарисовать любым доступным способом, например, так:

image1.width:=x\*20;

image1.Height:=y\*20;

image1.Picture.Bitmap.Width:=x\*20;

image1.Picture.Bitmap.Height:=y\*20;

with image1.Canvas do

begin

brush.Color:=clwhite;

fillrect(rect(0,0,width,height));

pen.Color:=clblack;

pen.Style:=psSolid;

pen.Width:=2;

Rectangle(1,1,x\*20,y\*20);

for j := 1 to y do

for i := 1 to x do

begin

moveto(i\*20,(j-1)\*20);

if cel[i,j].r=1 then lineto(i\*20,j\*20);

moveto((i-1)\*20,j\*20);

if cel[i,j].d=1 then lineto(i\*20,j\*20);

end;

Ну тут и объяснять ничего не надо.

Из особенностей этого алгоритма можно отметить высокую скорость работы – генерация матрицы размером 5000 на 5000 без отрисовки заняла всего 2 минуты, на моем не самом быстром домашнем ПК, а это 25 миллионов ячеек. Генерация 500х500 с отрисовкой – всего 6 секунд!

У меня нет возможности сравнить с другими алгоритмами, но там, в статье как раз на скорость и жаловались. Генерация маленьких лабиринтов – мгновенна. Стандартный размер 200х200 делает как раз за секунду.

Естественно рисовать таким способом большой лабиринт нет смысла – никакой памяти не хватит, можно делать частичную отрисовку на TImage и распечатывать частями. Но для лабиринтов размером до 300-400 ячеек на сторону подходит вполне.

Из недостатков стоит отметить как раз отсутствие привязки к точкам входа и выхода, а значит нет готового решения для тех, кто хотел бы иметь его сразу. Придется брать карандашик в руки и искать его самостоятельно, но в этом же ведь и суть любого лабиринта?

Еще недостаток – на мой взгляд, не слишком высокая сложность, но, если сделать хотя бы 15х15 уже сразу глазом не зацепишься за правильное решение.

Готовую программу выложил на Гитхабе, если кому интересно – можете скачать здесь. Хотелось бы услышать мнение специалистов.